TODAY: Strings
- tries & tryes
- compressed tries
- suffix trees & arrays
- document retrieval
- linear-time construction

String matching: given text $T$ & pattern $P$, find some/all occurrences of $P$ in $T$ as substrings
- one-shot: $O(T)$ time (Knuth, Morris, Pratt - SICOMP, 1977; Boyer & Moore - CACM, 1977; Karp & Rabin - IBM JRD)
- static DS: preprocess $T$, query $P$
  - goal: $O(P)$ query $O(T)$ space

- other data structures consider when $P$ has wildcards, or when $P$ need not match as an exact substring (Hamming/edit distance)
~ see e.g. [Cole, Gottlieb, Lewenstein - STOC 2004]
   [Maas & Novak - CPM 2005]
Warmup: predecessor among strings $T_1, \ldots, T_k$ (e.g. library search)

Trie = rooted tree with child branches labeled with letters in $\Sigma$

- to represent strings as root-to-leaf paths in a trie
  terminate them with a new letter $\$ \text{ (otherwise can't distinguish prefixes as absent or present)}$
  - e.g.: \%ana, ann, anna, anne \%

- in-order traversal of leaves = sorted strings

Trie representation: \( T = \# \text{ nodes in trie} \leq \sum_{i=1}^k |T_i| \)

- node stores children:
  1. as array
     \( \Rightarrow \text{blank cells store predecessor/successor} \)
  2. as balanced BST
     \( \Rightarrow \quad \text{query: } O(P) \quad \text{space: } O(T\Sigma) \)

- as hash table
  \( \Rightarrow \text{doesn't support predecessor queries/sorting} \)

- as van Emde Boas/\( y \)-fast
  \( \text{only need VEB when fall off} \quad O(P + \log \Sigma) \quad O(T) \)

\[ 3 + 3.5 = 3.75 \]
[Farach-Colton — personal communication, 2012]:

4. Node stores children:
   - as weight-balanced BST
   - \# descendant leaves in T \rightarrow \# leaves
   - Split children in left & right halves to optimally balance sum of weights
   - \Rightarrow every 2 edges followed either advances \( P \) letter or reduces \# candidate T strings to \( 2/3 \)
   - \Rightarrow charge to \( O(P) \) or \( O(lg k) \)

5. Leaf trimming (indirect):
   - Cut below maximally deep nodes with \( \geq |\Sigma| \) descendant leaves
   - \Rightarrow \# leaves in top trie \leq |T| / |\Sigma|
   - \Rightarrow \# branching top nodes \leq |T| / |\Sigma|
   - Use 4 on branching top nodes & 1 on top leaves (to find right bottom trie) & 2 on rest of top (\Rightarrow nonbranching in T)
   - \Rightarrow O(T) space on top
   - Bottom trees have \( < |\Sigma| \) descendant leaves
   - \Rightarrow 4 achieves \( O(P + lg \Sigma) \) query time

6. Suffix trays

   Simplification by Farach-Colton of:

   \[ O(P + lg \Sigma) \quad O(T) \]

   [Cole, Kopelowitz, Lewenstein — ICALP 2006]
Application: sorting strings $T_1, \ldots, T_k$ repeatedly insert into trie/tray
$\Rightarrow O(T + k \log \Sigma)$
- typically $O(T)$ & $\ll O(Tk \log k)$ via comparison

**Compressed trie:** contract nonbranching paths to single edge, keyed by first letter of path

e.g. {ana, ann, anna, anne}$^3$

- same representations apply, with $T = \#$ compressed nodes
**Suffix tree (trie):**

- compressed trie of all $|T|$ suffixes $T[i:]$ of $T$ (with $\$ $ appended)
- e.g.: $b a n a n a \$ 0 1 2 3 4 5 6$
- $|T| + 1$ leaves
- edge label = substring $T[i:j]$
- store as two indices $(i, j)$
- $O(T)$ space

**Applications:**

- search for $P$ gives subtree whose leaves correspond to all occurrences of $P$
  - $O(P)$ time via hashing
  - $O(P + \lg |T|)$ via tries \[\Rightarrow\] leaves sorted in $T$
  - $O(P + \lg \lg |T|)$ via hash + uEB \[\Rightarrow\]
- list first $k$ occurrences in $O(k)$ more time
  - every node points to leftmost descend. leaf
  - leaves connected via linked list
- # occurrences in $O(1)$ more time (Subtree sizes)

- longest repeated substring in $T$: $O(T)$ time
  - branching node of maximum “letter depth”
- longest substring match of $T[i:]$ vs. $T[j:]$: $O(1)$ via LCA query
- all occurrences of $T[i:j] = (j-i)^{th}$ "weighted" level ancestor of leaf for $T[i:]$ for compression
- store nodes in long path/ladder of $L15$ in van Emde Boas predecessor DS $\Rightarrow O(\log \log T)$
- can’t afford lookup tables at the bottom...
- use ladder decomposition on bottom trees
  $\Rightarrow$ jump to top of $O(\log \log n)$ ladders
  (to reach height $O(\log n)$)
- only need predecessor query on last ladder
  $\Rightarrow O(\log \log T)$ query & $O(T)$ space
  [Abbott, Baran, Demaine, ... – 6.897, Spr. 2005, L19.5]

- multiple documents via mult. $S$: $T = T_1 S_1 \cdots T_k S_k$
- count # distinct documents containing $P$
  - store # distinct $S$s below each node
- longest common substring in $O(T)$
  $\Rightarrow$ branching node with $\geq 2$ distinct $S$s below
- find $d$ distinct documents containing $P$ in $O(d)$ more
  "document retrieval problem" [Muthukrishnan–SODA 2002]
- each $S_i$ stores leaf # of previous $S_i$
- in interval $[l,n]$ of leaves below a node, want first $S_i$, i.e. $S_i$ storing $< l$ for each occ. $i$
- so find $m = \text{RMQ}(l,n)$ on array of stored values
- if stored value at leaf $m$ is $< i$:
  - found desired $S_i \sim$ output it
- recurse in intervals $[l,m-1]$ & $[m+1,n]$
  $\Rightarrow O(1)$ time per output (can stop anytime)
**Suffix arrays:** sort the suffixes of $T$ just store the indices $\Rightarrow O(T)$ space

- e.g. $b\ a\ n\ a\ n\ a\ \$  
  $\emptyset\ 1\ 2\ 3\ 4\ 5\ 6$

- searchable in $O(P\lg T)$ via binary search
- $lcp[i] = \text{length of longest common prefix of } i\text{th and } (i+1)\text{th suffix in order}$
- when binary searching in interval $SA[i:j]$, only need to compare from letter $RMQ_{lcp}(i,j-1)$
- via $RMQ$ of $L_{15}$, $O(P+\lg T)$ search [2007, PS4]

---

**Suffix trees $\leftrightarrow$ suffix arrays:**

- $\Rightarrow$ via in-order traversal of leaves
- $\Leftarrow$ via Cartesian tree of $lcp$ array
  - put all mins at root (unlike $L_{15}$)
  - nonleaf child subtrees: recurse
  - suffixes fit in between as leaves
  - $lcp$ value forming a node $= \text{letter depth of that node}$
  $\Rightarrow$ edge length = child $lcp$ - parent $lcp$
  $\Rightarrow$ can reconstruct labels
- all doable in linear time [L15]
- $lcp$s computable in $O(T)$ from SA [Kasai et al.-cmp 2001] or directly in suffix-array construction below
Constructing suffix array \( \Rightarrow \) tree \( \in O(T + \text{sort}(\Sigma)) \)

[\text{Kärkäinen \\& Sanders - ICALP 2003}], inspired by
[\text{Farach - FOCS 1997; Farach-Colton, Ferragina, Muthukrishnan - JACM}]

1. Sort \( \Sigma \) - initially in \( \text{sort}(\Sigma) \) time (or, if don't need children sorted, just number \( \Sigma \) arbitrarily)
   - later, radix sort in \( \Omega(T) \) time

2. replace each letter by its rank in \( \Sigma \) \( \Rightarrow \exists \leq |\Sigma| \)

3. form \( T_0 = \langle (T[3i], T[3i+1], T[3i+2]) \rangle \) for \( i = 0, 1, 2, \ldots \)
   \[ T_1 = \langle (T[3i+1], T[3i+2], T[3i+3]) \rangle \] for \( i = 0, 1, 2, \ldots \)
   \[ T_2 = \langle (T[3i+2], T[3i+3], T[3i+4]) \rangle \] for \( i = 0, 1, 2, \ldots \)

\( \Rightarrow \) suffixes(\( T \)) \( \approx \bigcup_{i=0}^{\leq |\Sigma|} \) suffixes(\( T_i \))

4. recurse on \( \langle T_0, T_1 \rangle \) \( \Rightarrow \frac{2}{3}|T| \) “letters”
   - sorted order & lcp's of \( \bigcup_{i=0}^{\leq |\Sigma|} \) suffixes(\( T_i \))

5. radix sort suffixes(\( T_2 \)) by writing
   \( T_2[i:] = T[3i+2:] \) \( \approx \langle T[3i+2], T[3i+3:] \rangle \) \( \approx \langle T[3i+2], T_0[i+1:] \rangle \)
   - also get lcp's in suffixes(\( T_2 \)): try to extend by 1

6. merge \( \bigcup_{i=0}^{\leq |\Sigma|} \) Suffixes(\( T_i \)) with suffixes(\( T_2 \)) via:
   - \( T_0[i:] \) vs. \( T_2[j:] \) = \( T[3i:] \) vs. \( T[3j+2:] \)
   
   \[ \langle T[3i], T[3i+1:] \rangle \] vs. \( \langle T[3j+2], T[3j+3:] \rangle \)

   - \( T_1[i:] \) vs. \( T_2[j:] \) = \( T[3i+1:] \) vs. \( T[3j+2:] \)
   
   \[ \langle T[3i+1], T[3i+2:], T[3i+3:] \rangle \] vs. \( \langle T[3j+2], T[3j+3:], T[3j+4:] \rangle \)

   - also get lcp's: try to extend by 1 or 2

\( \Rightarrow T(n) = T(\frac{2}{3}n) + O(n) = O(n) \) \( (n = |T|) \)